1. ZooKeeper 启动脚本分析

根据我以往看启动脚本的经验，这些.sh 脚本的最底层，一定是转到 某个Java 类去执行

```
1、最开始：zkServer.sh start
2、最后面：java org.apache.zookeeper.server.quorum.QuorumPeerMain
```

最后的入口：QuorumPeerMain.main()

```
QuorumPeer： zookeeper集群中，具备选举权和被选举权的一个节点，就被抽象为一个QuorumPeer
```

2. 具体的脚本 zkServer.sh 解析

```
# use POSIX interface, symlink is followed automatically
ZOOBIN="${BASH_SOURCE-$0}"
ZOOBIN="$(dirname "${ZOOBIN}")"
ZOOBINDIR="$(cd "${ZOOBIN}"; pwd)"

# 这里会先配置一些必要的环境变量
if [ -e "$ZOOBIN/../libexec/zkEnv.sh" ]; then
  . "$ZOOBINDIR"/../libexec/zkEnv.sh
else
  . "$ZOOBINDIR"/zkEnv.sh
fi

#此处配置JMX，默认是开启的，支持本地连接，如果远程连接，需要单独配置
# See the following page for extensive details on setting
# up the JVM to accept JMX remote management:
# http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html
# by default we allow local JMX connections
if [ "x$JMXLOCALONLY" = "x" ]
then
    JMXLOCALONLY=false
fi

if [ "x$JMXDISABLE" = "x" ] || [ "$JMXDISABLE" = 'false' ]
then

  #执行这里
  echo "ZooKeeper JMX enabled by default" >&2
  if [ "x$JMXPORT" = "x" ]
```

```
  then
    # for some reason these two options are necessary on jdk6 on Ubuntu
    # accord to the docs they are not necessary, but otw jconsole cannot do a
local attach

   #  作为 java 命令的参数： java命令： 启动JVM
    # 配置远程连接，修改ZOOMAIN变量
    # 配置远程连接，修改ZOOMAIN变量
    # 配置远程连接，修改ZOOMAIN变量
    ZOOMAIN="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.local.only=$JMXLOCALONLY
org.apache.zookeeper.server.quorum.QuorumPeerMain"
  else
    if [ "x$JMXAUTH" = "x" ]
    then
      JMXAUTH=false
    fi
    if [ "x$JMXSSL" = "x" ]
    then
      JMXSSL=false
    fi
    if [ "x$JMXLOG4J" = "x" ]
    then
      JMXLOG4J=true
    fi
    echo "ZooKeeper remote JMX Port set to $JMXPORT" >&2
    echo "ZooKeeper remote JMX authenticate set to $JMXAUTH" >&2
    echo "ZooKeeper remote JMX ssl set to $JMXSSL" >&2
    echo "ZooKeeper remote JMX log4j set to $JMXLOG4J" >&2
    ZOOMAIN="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=$JMXPORT -
Dcom.sun.management.jmxremote.authenticate=$JMXAUTH -
Dcom.sun.management.jmxremote.ssl=$JMXSSL -
Dzookeeper.jmx.log4j.disable=$JMXLOG4J
org.apache.zookeeper.server.quorum.QuorumPeerMain"
  fi
else
    # 运行主类
    # 运行主类
    # 运行主类
    echo "JMX disabled by user request" >&2
    ZOOMAIN="org.apache.zookeeper.server.quorum.QuorumPeerMain"
fi

if [ "x$SERVER_JVMFLAGS" != "x" ]
then
    JVMFLAGS="$SERVER_JVMFLAGS $JVMFLAGS"
fi

if [ "x$2" != "x" ]
then
    ZOOCFG="$ZOOCFGDIR/$2"
fi

# if we give a more complicated path to the config, don't screw around in
$ZOOCFGDIR
if [ "x$(dirname "$ZOOCFG")" != "x$ZOOCFGDIR" ]
then
```

```
    ZOOCFG="$2"
fi

if $cygwin
then
    ZOOCFG=`cygpath -wp "$ZOOCFG"`
    # cygwin has a "kill" in the shell itself, gets confused
    KILL=/bin/kill
else
    KILL=kill
fi

echo "Using config: $ZOOCFG" >&2

case "$OSTYPE" in
*solaris*)
  GREP=/usr/xpg4/bin/grep
  ;;
*)
  GREP=grep
  ;;
esac

# 从配置文件中，找出 dataDir 和 dataLogDir
ZOO_DATADIR="$($GREP "^[[:space:]]*dataDir" "$ZOOCFG" | sed -e 's/.*=//')"
ZOO_DATADIR="$(echo -e "${ZOO_DATADIR}" | sed -e 's/^[[:space:]]*//' -e
's/[[:space:]]*$//')"
ZOO_DATALOGDIR="$($GREP "^[[:space:]]*dataLogDir" "$ZOOCFG" | sed -e 's/.*=//')"

# iff autocreate is turned off and the datadirs don't exist fail
# immediately as we can't create the PID file, etc..., anyway.
if [ -n "$ZOO_DATADIR_AUTOCREATE_DISABLE" ]; then
    if [ ! -d "$ZOO_DATADIR/version-2" ]; then
        echo "ZooKeeper data directory is missing at $ZOO_DATADIR fix the path
or run initialize"
        exit 1
    fi

    if [ -n "$ZOO_DATALOGDIR" ] && [ ! -d "$ZOO_DATALOGDIR/version-2" ]; then
        echo "ZooKeeper txnlog directory is missing at $ZOO_DATALOGDIR fix the
path or run initialize"
        exit 1
    fi
    ZOO_DATADIR_AUTOCREATE="-Dzookeeper.datadir.autocreate=false"
fi
# 在类linux系统上，当zookeeper启动时，会创建一个用于存储进程id的zookeeper_server.pid文件
（在数据目录下），每次执行start、stop、restart命令时都会检查这个命令来判断zookeeper的启动状
态，而前台启动命令start-foreground不会检查这个文件，所以 start-foreground 和其他命令不是
一个系列，最好不混用。
# windows的zkServer.cmd 没有这个 进程文件 的操作
if [ -z "$ZOOPIDFILE" ]; then
    if [ ! -d "$ZOO_DATADIR" ]; then
        mkdir -p "$ZOO_DATADIR"
    fi
    ZOOPIDFILE="$ZOO_DATADIR/zookeeper_server.pid"
else
    # ensure it exists, otw stop will fail
    mkdir -p "$(dirname "$ZOOPIDFILE")"
```

```
fi

if [ ! -w "$ZOO_LOG_DIR" ] ; then
mkdir -p "$ZOO_LOG_DIR"
fi

ZOO_LOG_FILE=zookeeper-$USER-server-$HOSTNAME.log
_ZOO_DAEMON_OUT="$ZOO_LOG_DIR/zookeeper-$USER-server-$HOSTNAME.out"

# 正常启动的时候： zkServer.sh start/stop
# 按照命令不同来执行启动($1就是脚本的第一个参数，有可能是：start, stop, restart, status
等)
case $1 in

# start 命令（zkServer.sh start）
# start 命令（zkServer.sh start）
# start 命令（zkServer.sh start）
start)
    echo  -n "Starting zookeeper ... "
    if [ -f "$ZOOPIDFILE" ]; then
      if kill -0 `cat "$ZOOPIDFILE"` > /dev/null 2>&1; then
# 如果zookeeper_server.pid中存储的进程id代表的进程存在（kill -0 pid），说明zookeeper已
经启动了，无需再次启动，脚本退出执行
         echo $command already running as process `cat "$ZOOPIDFILE"`.
         exit 1
      fi
    fi
    # 其实最终转到使用 java 命令启动 ZOOMAIN 这个类  java $ZOOMAIN
    # 其实最终转到使用 java 命令启动 ZOOMAIN 这个类  java $ZOOMAIN
    # 其实最终转到使用 java 命令启动 ZOOMAIN 这个类  java $ZOOMAIN
    nohup "$JAVA" $ZOO_DATADIR_AUTOCREATE "-Dzookeeper.log.dir=${ZOO_LOG_DIR}" \
    "-Dzookeeper.log.file=${ZOO_LOG_FILE}" "-
Dzookeeper.root.logger=${ZOO_LOG4J_PROP}" \
    -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError='kill -9 %p' \
    -cp "$CLASSPATH" $JVMFLAGS $ZOOMAIN "$ZOOCFG" > "$_ZOO_DAEMON_OUT" 2>&1 <
/dev/null &
    if [ $? -eq 0 ]
# 如果上面的启动命令返回 0，说明执行成功，zookeeper启动了，就要把 进程id 写到
zookeeper_server.pid文件中
    then
      case "$OSTYPE" in
      *solaris*)
        /bin/echo "${!}\\c" > "$ZOOPIDFILE"
        ;;
      *)
        /bin/echo -n $! > "$ZOOPIDFILE"
        ;;
      esac
      if [ $? -eq 0 ];
      then
        sleep 1
        pid=$(cat "${ZOOPIDFILE}")
        if ps -p "${pid}" > /dev/null 2>&1; then
          echo STARTED
        else
          echo FAILED TO START
          exit 1
        fi
```

```
      else
        echo FAILED TO WRITE PID
        exit 1
      fi
    else
      echo SERVER DID NOT START
      exit 1
    fi
    ;;

# 前台启动命令
start-foreground)
    ZOO_CMD=(exec "$JAVA")
    if [ "${ZOO_NOEXEC}" != "" ]; then
      ZOO_CMD=("$JAVA")
    fi
    "${ZOO_CMD[@]}" $ZOO_DATADIR_AUTOCREATE "-Dzookeeper.log.dir=${ZOO_LOG_DIR}"
\
    "-Dzookeeper.log.file=${ZOO_LOG_FILE}" "-
Dzookeeper.root.logger=${ZOO_LOG4J_PROP}" \
    -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError='kill -9 %p' \
    -cp "$CLASSPATH" $JVMFLAGS $ZOOMAIN "$ZOOCFG"
    ;;

# 打印启动命令
print-cmd)
    echo "\"$JAVA\" $ZOO_DATADIR_AUTOCREATE -
Dzookeeper.log.dir=\"${ZOO_LOG_DIR}\" \
    -Dzookeeper.log.file=\"${ZOO_LOG_FILE}\" -
Dzookeeper.root.logger=\"${ZOO_LOG4J_PROP}\" \
    -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError='kill -9 %p' \
    -cp \"$CLASSPATH\" $JVMFLAGS $ZOOMAIN \"$ZOOCFG\" > \"$_ZOO_DAEMON_OUT\"
2>&1 < /dev/null"
    ;;

# 停止服务：会判断zookeeper_server.pid文件是否存在，如果不存在说明zookeeper没有运行
stop)
    echo -n "Stopping zookeeper ... "
    if [ ! -f "$ZOOPIDFILE" ]
    then
      echo "no zookeeper to stop (could not find file $ZOOPIDFILE)"
    else

# 停止服务就是执行两条命令：
# 第一条kill pid，注意没带信号变量，他会让进程清理善后工作，而不是直接杀死进程；
# 第二条rm zookeeper_server.pid，删除进程文件，也就是说zookeeper 访问没有启动的时候，是没
有这个文件的
      $KILL $(cat "$ZOOPIDFILE")
      rm "$ZOOPIDFILE"
      echo STOPPED
    fi
    exit 0
    ;;

# 重启服务：执行stop，等三秒，执行start
restart)
    shift
    "$0" stop ${@}
```

```
    sleep 3
    "$0" start ${@}
    ;;

# 查询状态
status)
    # -q is necessary on some versions of linux where nc returns too quickly,
and no stat result is output
    clientPortAddress=`$GREP "^[[:space:]]*clientPortAddress[^[:alpha:]]"
"$ZOOCFG" | sed -e 's/.*=//'`
    if ! [ $clientPortAddress ]
    then
   clientPortAddress="localhost"
    fi
    clientPort=`$GREP "^[[:space:]]*clientPort[^[:alpha:]]" "$ZOOCFG" | sed -e
's/.*=//'`
    if ! [[ "$clientPort"  =~ ^[0-9]+$ ]]
    then
       dataDir=`$GREP "^[[:space:]]*dataDir" "$ZOOCFG" | sed -e 's/.*=//'`
       myid=`cat "$dataDir/myid"`
       if ! [[ "$myid" =~ ^[0-9]+$ ]] ; then
         echo "clientPort not found and myid could not be determined.
Terminating."
         exit 1
       fi
       clientPortAndAddress=`$GREP "^[[:space:]]*server.$myid=.*;.*" "$ZOOCFG" |
sed -e 's/.*=//' | sed -e 's/.*;//'`
       if [ ! "$clientPortAndAddress" ] ; then
           echo "Client port not found in static config file. Looking in dynamic
config file."
           dynamicConfigFile=`$GREP "^[[:space:]]*dynamicConfigFile" "$ZOOCFG" |
sed -e 's/.*=//'`
           clientPortAndAddress=`$GREP "^[[:space:]]*server.$myid=.*;.*"
"$dynamicConfigFile" | sed -e 's/.*=//' | sed -e 's/.*;//'`
       fi
       if [ ! "$clientPortAndAddress" ] ; then
          echo "Client port not found. Terminating."
          exit 1
       fi
       if [[ "$clientPortAndAddress" =~ ^.*:[0-9]+ ]] ; then
          clientPortAddress=`echo "$clientPortAndAddress" | sed -e 's/:.*//'`
       fi
       clientPort=`echo "$clientPortAndAddress" | sed -e 's/.*://'`
       if [ ! "$clientPort" ] ; then
          echo "Client port not found. Terminating."
          exit 1
       fi
    fi
    echo "Client port found: $clientPort. Client address: $clientPortAddress."

# 这里会向 FourLetterWordMain 发送 srvr命令，从返回结果中 找到包含Mode的行，最终显示 该节
点是leader还是follower
    STAT=`"$JAVA" "-Dzookeeper.log.dir=${ZOO_LOG_DIR}" "-
Dzookeeper.root.logger=${ZOO_LOG4J_PROP}" "-Dzookeeper.log.file=${ZOO_LOG_FILE}"
\
             -cp "$CLASSPATH" $JVMFLAGS
org.apache.zookeeper.client.FourLetterWordMain \
             $clientPortAddress $clientPort srvr 2> /dev/null    \
```

```
            | $GREP Mode`
    if [ "x$STAT" = "x" ]
    then
        echo "Error contacting service. It is probably not running."
        exit 1
    else
        echo $STAT
        exit 0
    fi
    ;;

# 对于不认识的命令，打印帮助信息
*)
    echo "Usage: $0 [--config <conf-dir>] {start|start-
foreground|stop|restart|status|print-cmd}" >&2

esac
```